Math 415 Final Project: Application of Classical Cryptosystems to Music

Jake Gendreau

December 2, 2024

Contents

1 Abstract

This paper will explore classical cryptosystems and their adaptations to take in and spit out musical melodies. This topic comes from my interests in both music and mathematics. As the class has progressed, I have been making connections between the concepts in class and the music that I play. Through this paper, the same melodies will be applied to each of the cryptosystems in order to determine which makes the best sounding result, as measured by my ear.

2 Introduction

In typical applications of cryptosystems, the set of valid characters is the letters of the english alphabet. In order to adapt these systems for musical systems, I defined the set of valid characters to be $C = \{A, Bb, B, C, Db, D, Eb, E, F, Gb, G, Ab, /\}$. Each symbol represents a note of 12 tone equal temperament, and '/' represents rest. This set contains 13 characters. For simplicity's sake, I will restrict the plaintext melodies to be a single measure of eight eighth notes.

In my discoveries, I discuss the observed relationships between the melodies and keys fed into the ciphers, and the resulting melody. I use musical terminology, which I will explain here:

- Key (music) - The set of pitches that will be used in a tune. Each key has a corresponding major and minor version.
- Scale - An ascending line that plays every note of a musical key.
- Diatonic - Uses the tones of a scale.

- Non-diatonic - Does not use the tones of a scale.
- Chord - Pitches in a key that are played to create harmony.
- Chord extensions - Additional notes in a chord that give it a different texture

3 Modifying Cryptosystems to Work With Music

3.1 Caesar Cipher

3.1.1 Standard Usage

The standard usage of the Caesar Cipher is as follows:

1. Choose a key, k, from 0 - 26.
2. Shift each token in the plaintext by the key, then take it modulus 26.

[TW06]

3.1.2 Modification to Work With Music

For the Caesar Cipher, it is trivial to make it work with music. The following must be changed:

- The key, k, must be 0 - 13 instead of 0 - 26.
- The modulus of each shift must be 13 instead of 26.

3.1.3 Example

1. Choose the plaintext {D, E, F, G, E, /, C, D} and they key $k = C$.
2. Shift each key by C modulus 13.
3. The resulting ciphertext is {F, G, Ab, A, G, B, Eb, F}

3.2 Vigenere Cipher

3.2.1 Standard Usage

The standard usage of the Caesar Cipher is as follows:

1. Choose a passphrase.
2. Repeat the passphrase enough times so that it is equal to the length of the plaintext, truncating if needed.
3. Combine the plaintext with the passphrase token-wise, modulus 26.

[TW06]

3.2.2 Modification to Work With Music

To make the Viginere Cipher work with music, it remains trivial. The following must be changed:

- The key must be another melody, not a single note or character.
- The modulus of each token-wise combination must be taken mod 13, not mod 26.

3.2.3 Example

1. Choose plaintext = {D, E, F, G, E, /, C, D}.
2. Choose passphrase = {A, B, C}.
3. Expand the passphrase = {A, B, C, A, B, C, A, B}.
4. For each token in the plaintext, combine it with the corresponding token in the passphrase, mod 13.
5. The resulting ciphertext is {D, Gb, Ab, G, Gb, B, C, E}.

3.3 Affine Cipher

3.3.1 Standard Usage

The standard usage of the Affine Cipher is as follows:

1. Choose (α, β) such that $gcd(\alpha, 26) = 1$, and $\beta \in \mathbf{Z}$
2. For each token in the plaintext, the corresponding ciphertext token is equal to $\alpha * x + \beta(mod 26)$

[TW06]

3.3.2 Modification to Work With Music

- The property must be retained that $gcd(\beta, n) = 1$, where n is the modulus. Since the new modulus is 13, this will always be the case.
- When doing the operation $\alpha * x + \beta(mod n)$, n must be equal to 13, not 26.

3.3.3 Example

1. Choose plaintext = {D, E, F, G, E, /, C, D}.
2. Choose key $(\alpha, \beta) = (Db, G)$.
3. For each token in the plaintext, perform the operation $c = \alpha * x + \beta(mod 13)$, where c is the corresponding ciphertext token.
4. The resulting ciphertext is {Db, /, C, Ab, /, Eb, Gb, Db}.

3.4 Hill Cipher

3.4.1 Standard Usage

1. Choose an n that will define the dimension of the encryption matrix.
2. Create an $n * n$ invertible matrix, M, using the values 0 - 25.
3. Split the plaintext up into m segments of size n, appending 'x' if needed.
4. For each of the m plaintext segments, multiply the vector by the matrix. Append the encrypted segment to the ciphertext.

[TW06]

3.4.2 Modification to Work With Music

- M must consist of the values 0 - 12, instead of 0 - 25.
- The property that M must be invertible mod n must be retained. Since $n = 13$ is prime, it is sufficient that $det(M) \neq 0$.
- Instead of appending 'x', append '/' if needed.

3.4.3 Example

1. Choose plaintext = {D, E, F, G, E, /, C, D}.
2. Choose encryption matrix

$$M = \begin{bmatrix} Ab & A & B \\ C & D & E \\ F & G & A \end{bmatrix}$$

   Since $det(M) \equiv 3 \pmod{13}$, this matrix is invertible, and can be used.

3. Split the plaintext into vectors of size 3. {D, E, F}, {G, E, /}, {C, D, /}. In this case, a '/' had to be appended.
4. Multiplying each of these vectors by M results in {Eb, B, Eb}, {Db, Eb, E}, {D, Bb, Gb}.
5. Combining these vectors gives the ciphertext {Eb, B, Eb, Db, Eb, E, D, Bb, Gb}.

3.5 Playfair Cipher

3.5.1 History

The Playfair Cipher was created by Charles Wheatstone, but carries the name of his friend - Lyon Playfair. Playfair was "A scientist and public figure of Victorian England..." [Kah67]. As such, he had some pull within the English government. In January of 1854, Playfair demonstrated "Wheatstone's newly discovered symmetrical cipher" at a dinner hosted by Lord Granville, which included guests such as the Home Secretary and the future Prime Minister [Kah67].

Wheatstone and Playfair explained the cipher to some higher-ups in the government, where it was eventually accepted. It may have been used in the Crimean War, but there is no solid evidence of it. It was, however, defnitely used during the Boer war [Kah67].

3.5.2 Standard Usage

1. The first step is to set up the encryption matrix:
   (a) Choose a passphrase.
   (b) Remove duplicate characters from the passphrase.
   (c) Fill in the first n positions of a 5x5 matrix using the passphrase, where n is the length of the passphrase without duplicate characters.
   (d) Fill in the rest of the 5x5 matrix with the characters of the alphabet, treating I and J as the same character.
2. The second step is to set up the plaintext such that it is ready for encryption:
   (a) Between any pair of double letters, insert an 'x'.
   (b) If needed, append an 'x' onto the end so that the length of the plaintext is even.
   (c) Divide the plaintext into groups of two letters each.
3. The last step is to encrypt using the following scheme:
   (a) If the two letters in the plaintext group are not in the same row or column of the encryption matrix, replace each letter by the letter that is in its row and the column of the other letter.
   (b) If the last two letters are in the same row, replace each letter with the letter immediately to its right, with the matrix wrapping around from the last column to the first.
   (c) If the two letters are in the same column, replace each letter with the letter immediately below it, with the matrix wrapping around from the last tow to the first.

[TW06]

3.5.3 Modification to Work With Music

1. Set up encryption matrix:
   (a) Choose a passphrase.
   (b) Remove duplicate pitches from the passphrase.
   (c) Fill in the first n positions of a 3x4 matrix using the passphrase, where n is the length of the passphrase without duplicate pitches.
   (d) Fill in the rest of the 3x5 matrix with the pitches in an octave, treating '/' and 'A' as the same pitch.
2. Set up the plaintext for use:
   (a) Between any pair of double pitches, insert a '/'.
   (b) If needed, append a '/' onto the end so that the length of the plaintext is even.
   (c) Divide the plaintext into groups of two pitches each.
3. Encryption: The process of encryption remains the same

3.5.4 Example

The plaintext {D, E, F, G, E, /, C, D} would get encrypted to {E, Ab, Gb, Ab, Bb, D, D, A}

3.6 ADFGX Cipher

3.6.1 History

The ADFGX Cipher was the result of many candidate ciphers being put into competition at a German conference. It was used during World War I by the German army, and was the most impossible cipher in existence at the time. Georges Painvin was the one who would eventually end up cracking it. For three weeks, Painvin struggled at his desk, trying every possible angle of attack, until he finally figure it out [Kah67].

As time progressed, Painvin got faster and faster at decrypting German messages. Eventually, it would take him less than 24 hours to decrypt incoming messages. Because of Painvin's efforts, the French were able to prepare for various German attacks, which saved lives and helped to win the war [Kah67].

3.6.2 Standard Usage

1. The first step of using the ADFGX Cipher is to set up the encryption matrix.
   (a) Insert every character of the alphabet into a 5x5 matrix in random order, treating 'I' and 'J' as a single character.

(b) Label the columns and rows of the matrix with A, D, F, G, and X. An example would be

$$\begin{array}{c|ccccc} & A & D & F & G & X \\ \hline A & A & B & C & D & E \\ D & F & G & H & I & K \\ F & L & M & N & O & P \\ G & Q & R & S & T & U \\ X & V & W & X & Y & Z \end{array}$$

It is important to note that the values of the matrix do not have to be in the order of the alphabet, they could be in any order.

2. The next step is to encrypt the plaintext.

   (a) Take each letter of the plaintext, and replace it with the row and column values of its position in the matrix. For example, H would be encrypted to DF.

   (b) Decide a keyword that has all unique letters. Organize the letters in the partially encrypted plaintext in a matrix under the keyword. For example, using the matrix above, a plaintext of "HELLO", and a keywordof "MATH", the new matrix would be:

$$\begin{array}{cccc} M & A & T & H \\ \hline D & F & A & X \\ F & A & F & A \\ F & G & & \end{array}$$

   (c) Now, reorder the columns such that the column labels are in alphabetic order:

$$\begin{array}{cccc} A & H & M & T \\ \hline F & X & D & A \\ A & A & F & F \\ G & & F & \end{array}$$

   (d) Lastly, the ciphertext is obtained by reading down the columns. In this example, it would be FAGXADFFAF.

[TW06]

3.6.3 Modification to Work With Music

By changing the labels on the outside of the matrix, I am able to decide which notes end up in the ciphertext. Given a 4x3 matrix, I have 7 possible pitches. Taking 1 away for the rest leaves me with just enough to make a blues scale.

1. The creation of the matrix:

   (a) Insert the pitches from an octave of 12 tone equal temperament into a 4x3 matrix, treating 'A' and '/' as the same pitch.
   (b) Label the rows with 'C', 'Eb', and 'F'.
   (c) Label the columns with 'Gb', 'G', 'Bb', and '/'.

   The result should look similar to:

	Gb	*G*	*Bb*	*/*
C	*A*	*Bb*	*B*	*C*
Eb	*Db*	*D*	*Eb*	*E*
F	*F*	*Gb*	*G*	*Ab*

   The encryption matrix is now complete.

2. Encryption:
   - The passkey should be a melody, not a phrase.
   - The columns won't be sorted alphabetically. Instead, they will be sorted by scale degree ('A' is first, '/' is last).

3.6.4 Example

The plaintext {D, E, F, G, E, /, C, D}, encrypted with the key {A, B, C, D} would result in the ciphertext {Eb, F, Eb, C, G, Gb, /, /, Eb, F, C, Eb, /, Bb, Gb, G}

4 Discoveries

I used three different melodies to test my ciphers. The three are:

1. An A minor scale - {A, B, C, D, E, F, G, /}.
2. "The Lick", a famous jazz lick - {D, E, F, G, E, / , C, D}.
3. The note C repeated 8 times - {C, C, C, C, C, C, C, C}.

I tested all three melodies with each of the three cryptosystems. Unsurprisingly, most of the results were bad. There were, however, some exceptions.

4.1 Caesar Cipher

The Caesar Cipher sounded good with most melodies since the same note relationships would be held, but they would be shifted up by a consistent amount. This means that whatever I fed it, would more or less, persist after the encryption.

When I was making this Cipher, I had the vision of musical transposition in my head. This musical transposition, however, is not quite the same as a mathematical transposition. In musical transposition, rests are maintained, and only pitches are moved. However, with my implementation, the rests are treated as any other note, so they will be transposed as well.

4.2 Vigenere Cipher

The Vigenere Cipher could also sound good, but it depended on what I used for the key. I found that using the notes in a chord as the key tended to make the Vigenere Cipher sound good.

Specifically, I found that using a key of {A, C, E} or {C, E, G} - the A minor triad and C major triad, which is its corresponding major musical key - sounded good. It didn't really matter how many chord extensions I used, as long as the key was part of the chord signature of A minor, it would result in something good.

4.3 Affine Cipher

The Affine Cipher could also sound good when given the right key. If I were to use a key (α, β) such that α and β were the third and the seventh of the chord, it sounded better.

For example, with the melody that I have used frequently in this paper - {D, E, F, G, E, /, C, D} - the musical key is D minor. The third and the seventh of D minor would be F and C, respectively. Going through and applying the algorithm to each pitch with the key (F, C) yields {Db, E, B, D, E, F, Bb, Db}, which, interestingly, fits almost entirely into the key of F - the relative major to D minor.

Interestingly, this property holds true for all sorts of melodies and keys. Testing it with the A minor scale, along with the corresponding key yields {G, C, Eb, /, D, G, Bb, E}, which fits almost entirely into the key of C major, which is A minor's relative major.

4.4 Hill Cipher

The only scenario where I was able to get the Hill Cipher to sound good was when I encrypted with $3 * 3$ matrix such as:

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & A & / \end{bmatrix}$$

I believe this to be the case because all of the non-diatonic parts of the matrix - like {A, B, C} - of the encryption matrix lie in the rows, whereas the nice chordal parts lie in the colums - like {A, D, G}. Looking at how multiplication of vectors and matrices is done, the values of each row get multiplied by the vector values of the vector, then added together, whereas the columns are each individual. This leads to the non-diatonic parts combining, and potentially, canceling out.

4.5 Playfair Cipher

The Playfair cipher, similar to the Vigenere Cipher, tended to work nicely if the encryption key spelled out the chord of the musical key. For example, encrypting the A minor scale with the A minor chord resulted in a surprisingly beautiful melody.

Encryption with other chords, however, tended to make the resulting melody more non-diatonic the further away from the musical key center the encryption key is. For example, the A minor scale encrypted with the Bb minor triad tended to still sound good, whereas encryption with the E minor triad sounded worse.

4.6 ADFGX Cipher

The ADFGX Cipher was the best sounding algorithm. Because of how I modified the algorithm to work with music, I was able to choose the seven possible notes that the algorithm could output. I chose to make it the A minor blues scale. Since the output of the algorithm was limited to {A, C, D, Eb, F, G, /}, it tended to sound decent, no matter what I put into it.

That being said, as with many of the other algorithms, it tended to sound better if the encryption key that was fed to the algorithm was also a part of the musical key. The major triad sounded good, however, I found the minor triad with the major seventh extension to sound really good. For example, encrypting the A minor scale with the key {A, C, E, G} produced a very interesting melody, with a neat chromatic ascension at the beginning.

5 Computer Implementation

For the most part, the computer implementations are identical to the algorithms presented in the textbook. When the encryption function is called, the plaintext has already been gotten, validated, and tokenized.

5.1 Caesar Cipher

1. Get the key in the form of a note (e.g. Ab, C, /).
2. For each token in the plaintext, shift it by the key and add the shifted token to the ciphertext.
3. Print and return the ciphertext.

5.2 Vigenere Cipher

1. Get, validate, and tokenize the key in the form of a passphrase melody (e.g. {A, B, C}.
2. For each token in the plaintext, do the following, shift it according to the corresponding position in the passphrase, cycling through as needed.
3. Append the new ciphertext token to the ciphertext list.
4. Print and return the ciphertext.

5.3 Affine Cipher

1. Get, validate, and tokenize alpha.
2. Convert alpha to an integer representation of its token.
3. Get, validate, and tokenize beta.
4. Convert beta to an integer representation of its token.
5. For each token in the plaintext, multiply it by alpha, then add beta (mod 13).
6. Append the new token to the ciphertext.
7. Print and return the ciphertext.

5.4 Hill Cipher

1. Get the dimension of the encryption matrix from the user. Store the result as n.
2. Get the passphrase in the form of an $n * n$ token melody.
3. Create an empty $n * n$ array.
4. Insert each token from the passphrase into the matrix, upper left to bottom right.
5. Create 2D array with n empty arrays.
6. Split the plaintext into m vectors of size n. If needed, append '/' to get the last vector to size n.
7. Multiply each vector by the encryption matrix, and append each token of the vectors after multiplication to the ciphertext.
8. Print and return the ciphertext.

5.5 Playfair Cipher

1. Get, validate, and tokenize the key as a melody.
2. Remove duplicate entries from the melody.
3. Insert the tokens from the key into the first n positions of the matrix, upper left to bottom right.
4. Fill the remaining positions of the matrix with the valid tokens not in the key.
5. Split the plaintext into pairs, inserting '/' as needed.

6. For each pair:
    (a) Get the row and column of each token in the pair.
    (b) If the two have the same row, append the token to the right to the ciphertext for both values.
    (c) If the two have the same column, append the token to the bottom to the ciphertext for both values.
    (d) If they have different rows and columns, append the token on its row and the column of the other token to the ciphertext.
7. Print and return the ciphertext.

5.6 ADFGX Cipher

1. Set up the encryption matrix

$$M = \begin{array}{c|cccc} & Gb & G & Bb & / \\ \hline C & A & Bb & B & C \\ Eb & Db & D & Eb & E \\ F & F & Gb & G & Ab \end{array}$$

2. For each token in the plaintext, get its rown and column, append this pair to an array of tuples called pitchPairs.
3. Set up a 2D array that is the length of passTokens, where the first value of each array is the corresponding token in passTokens.
4. Insert each token of the pairs in pitchPairs into the 2D array.
5. Sort the arrays in the 2D array by scale degree (e.g. A is 0, '/' is 12).
6. For each array, ignore the first value (this is used for sorting), and insert the rest of the values into the ciphertext.
7. Print and return the ciphertext.

Bibliography

[Kah67] David Kahn. *The Codebreakers.* The Macmillan Company, 1967. ISBN: 0684831309.

[TW06] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory.* Pearson Education International, 2006. ISBN: 0131862391.